回路自動合成のための推論機構(計算アルゴリズムの基礎理論)

AUTHOR(S):

岩沼, 宏治; 原尾, 政輝

CITATION:

岩沼, 宏治 ...[et al]. 回路自動合成のための推論機構(計算アルゴリズムの基礎理論). 数理解析研究所講究録 1987, 625: 276-286

ISSUE DATE:

1987-05

URL:

http://hdl.handle.net/2433/99942

数理解析研究所講究録
625巻 1987年 276-286

回路自動合成のための推論機構

原尾 政輝　　岩沼 宏治
山形大学工学部 情報工学科

1．まえがき

回路を設計者にとって記述し易く簡潔な高レベルの言語で仕様記述し，その記述から詳細回路図レベルへ自動変換する設計自動化手法が重要な課題となっている[11]。本稿では、高レベル記述として再帰方程式，回路モデルとしてデータフロー演算に基づく関数ネットワーク[1]（FN）を用い,回路自動合成問題を再帰方程式からＦＮへの変換としてとらえ、人工知能的手法を用いた回路自動合成のための推論機構について考察する。まず、ＦＮをモデルとする回路実現言語を定義し、その様相論理に基づいた記述言語の実現について述べる。さらに、回路記述を知識工学的観点から行い、再帰方程式から回路の自動合成を、様相論理の枠組みでの定理証明システムとして定式化する。そこで推論機構の要である導出原理が、様相論理式から一階述語論理へのコンパイレーションの概念を用いて実現出来る事を示す。また処理の効率化のための知識の階層化やメタ推論などの機能についても述べる。

2．回路仕様記述言語

回路の仕様記述言語としては、設計者にとって分かり易くかつ簡潔で、記述の汎用性や記述の厳密さの条件を満たしていることが望ましい。再帰方程式系は、数学的に厳密に定義された記法であり計算論的にも万能なクラスを特性化する。また、再帰的表現は直観的にも理解しやすいし表現も簡潔である。このような理由からここでは回路仕様記述言語として再帰方程式を採用する。

2．1　再帰方程式

表現のメタ記号として "(","）","{","}"等を用いる。また、値変数を x,y,z,・・・で、関数定数を f,g,h,・・・で、述語定数を p,q,r,・・・で、関数変数を F,G,H,・・・で、述語変数を P,Q,R,・・・で表すことにする。特に、関数変数記号を**再帰記号**、それ以外のものを**自明な記号**と呼ぶ　再帰方程式とは、以上の記法を用いてフォーマルには次のように定義される。そこで，Ｆはm変数関数記号、ｅは再帰表現、＜＝ は代入を表すメタシンボルとする：
$F(x_1, \ldots, x_m) <= e.$

定義 2.1　回路の仕様記述とは、再帰方程式の有限集合で次の形のものである：

$$z = e(x,F) \text{ where } \quad F_1(\underline{x}) <= e_1(\underline{x},\underline{F}),\quad F_2(\underline{x}) <= e_2(\underline{x},\underline{F}),\quad \ldots\ldots\quad F_n(\underline{x}) <= e_n(\underline{x},\underline{F}).$$

ここで $\underline{x} = \{x_1, x_2, \ldots, x_m\}$, $\underline{F} = \{F_1, F_2, \ldots, F_n\}$で ｅ は $\underline{x}$ の上の表現である。

2．2 再帰スキーマ

再帰方程式でその記号が解釈されていないものを、**再帰スキーマ**と呼ぶ。

例 2.1 基本的なスキーマの例を次に示す。

schema_1:$F(x)\Leftarrow p(x)\rightarrow f(x),F(g(x))$

schema_2:$F(x)\Leftarrow p(x)\rightarrow G(x),F(f(x))$,
$G(x)\Leftarrow q(x)\rightarrow H(x),G(g(x))$,
.....
$H(x)\Leftarrow r(x)\rightarrow c(x),H(h(x))$

schema_3:$F(x)\Leftarrow p(x)\rightarrow f(x)$,
$h(F(g(x)))$

schema_4:$F(x)\Leftarrow p(x)\rightarrow f(x),h(x,F(g(x)))$

schema_5:$F(x)\Leftarrow p(x)\rightarrow f(x),h(g_1(x),F(g_2(x)))$

schema_6:$F(x,y)\Leftarrow p(x)\rightarrow f(x,y),h(x,F(g(x),y))$

schema_7:$F(x,y)\Leftarrow p(x)\rightarrow f(x,y),h(y,F(g_1(x),g_2(x)))$

schema_8:$F(x)\Leftarrow p(x)\rightarrow f(x),F(h(F(g(x))))$

schema_9:$F(x)\Leftarrow p(x)\rightarrow f(x),F(h_1(F(h_2(F(h_3(x))))))$

schema_10:$F(x)\Leftarrow p(x)\rightarrow f(x),F(h(x,F(g(x))))$

schema_11:$F(x)\Leftarrow p(x)\rightarrow f(x),h(F(g_1(x)),F(g_2(x)))$

3． 回路モデル

本章では、データフロー的制御を基にした回路のモデルであるＦＮ（関数ネットワーク）をまず定義する。次いで、ＦＮの動作記述を計算列の概念を導入して行う。この計算列の概念は、4章で回路実現言語の意味記述と結びつけられる。

3．1 関数ネットワーク

関数 $y = f(x_1,x_2,\ldots,x_n)$ や 述語 $y = p(x_1,\ldots,x_n)$に対し、入力端子 $\{x_1,\ldots,x_n\}$,出力 端子 y をもつ素子を対応させ、それを**関数回路**，**述語回路**と呼ぶ。

定義 3.1 再帰方程式の基本関数および基本述語に対応する機能を持った回路を関数素子、述語素子、その集合を基本素子とよびΩで表す。

特に制御用の基本素子として、単位時間の遅延素子reg(ﾚｼﾞｽﾀ) 、マルチプレクサ muxなどの素子をΩに含める。Ωがどのような素子を含むかによって、実現できる回路のクラスも異なってくる。前に述べたように、組合わせ素子だけではすべての再帰方程式に対応するＦＮは構成できない。従って、複雑なデータ構造をもったカウンタ、スタックを基本素子として用いる事もある。 一般のＦＮ回路はこのようなΩに含まれる基本回路素子を次の直・並列合成規則に従って組合わせて構成される。

(1) **直列合成**： $[f\&g](x) = f(g(x))$

(2) **並列合成**： $[f_1\#f_2](x) = [f_1(x),f_2(x)]$

(3) **選択接続**： $\pi_i\ (x_1,\ldots,x_n) = x_i$.

定義 3.2 回路の基本素子の集合をΩとする。Ωの要素を上述の規則に従って組合わせて構成される回路網を，Ωにより構成される**関数ネット**(Functional Net)と呼ぶ.

3．2 ＦＮの動作記述

ＦＮの回路動作はデータフロー演算に基づいていると仮定する。即ち、各素子はクロック信号と総ての入力が揃えば演算を実行し次の端子に出力する。ここでは、詳細なタイミングや信号制御については考慮せず、各素子の演算時間は0で、reg素子が回路全体の同期をとるとする。

定義 3.3 端子 X における計算列とは、時間 t から X の値域Dへの関数 s である。すなわち、ωを自然数の集合とすると　s：X×ω → D　である。

特に、⊥を未定義記号として$\Sigma = D \cup \{\perp\}$とおく。端子 X の計算列とは、端子 Xの信号値の時系列とみなすことが出来る。図3.1の各回路の各端子の計算列は次のようになる。　但し、x_{i0}は初期値としx_{it}, y_tで$s(X_i,t), s(Y,t)$を表わすものとする。

(1) $Y = f(X_1,X_2,\ldots,X_n)$の時（図3.1(a)）：

$s(X) = \langle x_{i0},\perp,\perp,\perp,\perp,\perp,\ldots.\rangle \in \Sigma^w$,　$s(Y) = \langle y_0,\perp,\perp,\perp,\perp,\perp,\ldots.\rangle \in \Sigma^w$

但し、$Y_0 = s(Y,0) = f(s(X_1,0), s(X_2,0), \ldots, s(X_n,0))$

(2) $Y = reg(X)$ の時（図3.1(b)）：

$s(X) = \langle x_0,\perp,\perp,\perp,\perp,\perp,\ldots.\rangle \in \Sigma^w$,　$s(Y) = \langle \perp,x_0,\perp,\perp,\perp,\perp,\perp,\ldots\rangle \in \Sigma^w$

(3) $Y = f(X), X = reg(Y)$ の時（図3.1(c)）：

$s(X) = \langle x_0,x_1,x_2,x_3,x_4,x_5,\ldots.\rangle \in \Sigma^w$, $s(Y) = \langle \perp,y_0,y_1,y_2,y_3,y_4,,\ldots.\rangle \in \Sigma^w$

但し、$y_i = f(x_i)$とする。

以後、混乱が生じない時は、⊥だけからなる無限部分系列は空系列と見做し、端子Xの計算列s(X)とΣ^+の元を同一視する。回路の同期はreg素子でとられ、図3.3(c)のようなループを含む回路の計算列は無限系列になる。

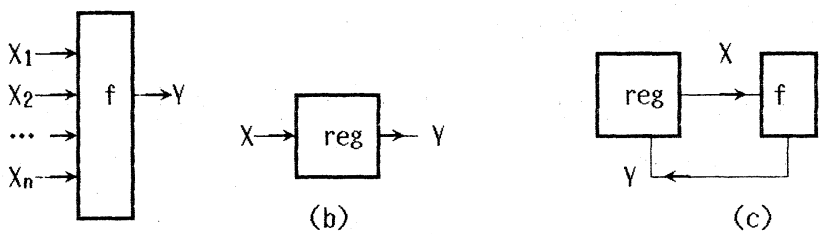


図3.1 回路と計算列

定義 3.4 計算列 s(X) と s(Y) が等価とは、すべての t に対して　s(X,t) = s(Y,t) が成立つことである。また、計算列上のnext演算子○とは、次のように定義される：　○s(X,t) = s(X,t+1).即ち、○Xは時間軸を未来に１つずらした計算列である。

命題 3.1 回路素子　$Y = f(X_1,X_2,\ldots,X_n)$ の各端子の計算列を $s(Y), s(X_1), \ldots, s(X_n)$とすると、次の関係：$s(Y) = f(s(X_1), s(X_2), \ldots, s(X_n))$.が成立つ。特に、$Y = reg(X)$ の場合は次の関係：　$s(Y) = ○s(X)$が成立つ■

記法：以後、混乱が生じない限りs(X)などをsを省略して単に X と書き、端子名は計算列をも表わすものとする。ＦＮで計算列 X,Y に関係　$Y = f(X)$, $○X = Y$ が成立つ時、この計算列間の関係を［$○X = f(X)$］で表わし、これを**ＦＮの回路実現**［$○X = f(X)$］と呼ぶ。

ＦＮ回路で次の機能：　if p(X) then Z = X else Y = f(X) をもつ標準回路（図3.2）を考える。ここで、P(X) は述語に相当するゲート回路である。p(X,t) = false　とすると、X,Z の計算列は　$s(X) = \langle x_0,x_1,x_2,\ldots,x_{t-1},\perp,\perp,\ldots\rangle, s(Z) = \langle \perp,\perp,\perp,\ldots,\perp,x_t,\perp,\perp,\ldots\rangle$

となる。このＦＮを回路実現言語では [○X = f(X)]@p(X)と書く。この表現では端子の結合を表に出して記述している。従ってこの表現から回路レイアウトが可能になる。

定理 3.1 pを自明な述語、r,t1,...,tn を自明な表現とする。このとき次のスキーマ

$F(X_1,...,X_n) <= p \rightarrow r,F(t_1,...,t_n)$

はつぎの回路実現を持つ：

$[\ \bigcirc X_1 = g_1(X_1,...,X_n),$
$\bigcirc X_2 = g_2(X_1,...,X_n),$
$.....$
$\bigcirc X_n = g_n(X_1,...,X_n)\ ]@p(X_1,...,X_n)$

但し、$t_i = g_i(X_1,...,X_n), r = f(X_1,...,X_n)$ とする。

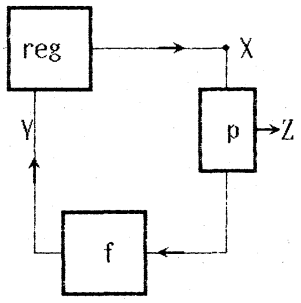


図3.2 標準回路

定理 3.2 再帰方程式　schema_2:

$F(X)<= p(X) \rightarrow G(X),F(f(X)),$
$G(X)<= q(X) \rightarrow H(X),G(g(X)),$
$.....$
$H(X)<= r(X) \rightarrow c(X),H(h(X)).$

は次の回路実現をもつ：

$[\ \bigcirc X_1 = f(X_1)]@p(X_1),\ [\ \bigcirc X_2 = g(X_2)]@p(X_2),\ \ ...,$
$[\ \bigcirc X_n = h(X_n)]@p(X_n),\ \ y = c(X_n).$

schema_1 やschema_2はこのように素直にＦＮに対応し、具体的な関数や述語回路をスキーマにあてはめてその回路を実現することができる。その他のスキーマは、それぞれのスキーマから schema_1 またはschema_2に変換することによって同様に回路実現を与えることが出来る。

4　回路実現言語

本章では、前章で定義した回路実現の意味を忠実に反映する言語を設計する。回路は関数型言語とも親和性があることが知られているが、本文ではまず回路の自動合成の枠組みをつくるため、第二に回路の表現には値，位置更には時間の情報も陽に表現出来る事が望ましいこと、検証手法の点からは，定理証明的手法が素直に導入出来るものが望ましいこと等より，論理型言語を採用した。しかし、ＦＮの意味記述には一階述語論理では本質的に無理な所があるので、ここでは様相論理を基礎にする。

4．1　様相論理

様相述語論理は、普通の一階述語論理に幾つかの様相演算子を導入して構成される。ここでは、時相論理で用いられるnext演算子○および通常のalways演算子□を基礎にする。まず様相論理記号の直観的意味は次の通りである。A がある述語論理式のとき

(a) ○A ：次の時刻にA は true である。
(b) □A ：すべての時刻に A はtrue である。
(c) ◇A ： A がtrue になる時刻がある。

ここで用いる論理体系は文献[8]と本質的には同一なのでその詳細は割愛する。ただ様相述語論理と回路動作との関係を示すため、この論理のモデルについて簡単に述べる。モデルとは3項組（D，S，α）である。ここで、Dはデータ領域，Sは状態集合，αは意味解釈関数。いま、f，p をそれぞれk-変数関数、k-変数述語とするとαは次のような写像である：

$$\alpha(f)\in D^k \to D, \qquad \alpha(p)\in D^k \to \{true,false\},$$

元 $s\in S^+$は状態の系列で、$s = <s_0,s_1,s_2,\ldots,s_{k-1},.>$の時解釈αはそれぞれの状態$s_i$によって決定される。s および s_i での解釈をα_s,α_iとそれぞれ略記する。これを用いて論理式の解釈は次のようになる：

(1)$\alpha_s[X]=\alpha_1[X]$,　(2)$\alpha_s[f(t_1,\ldots,t_n)]=\alpha(f)(\alpha[t_1],\ldots,\alpha[t_n])$,
(3)$\alpha_s[p(t_1,\ldots,t_n)]=\alpha(p)(\alpha[t_1],\cdots,\alpha[t_n])$　(4)$\alpha_i[\bigcirc t]=\alpha_{i+1}[t]$,
(5)$\alpha_s[A]$=true iff $\forall i,\alpha_i[A]$=true,　(6)$\alpha_s[\Box A]$=true iff $\forall i,\alpha_i[A]$=true,
(7)$\alpha_s[\bigcirc A]$=true iff $\forall i,\alpha_i[\bigcirc t]$=true,
(8)$\alpha_s[A\wedge B]$=true iff $\alpha_s[A]$=true and $\alpha_s[B]$=true
(9)$\alpha_s[\sim A]$=true iff $\alpha_s[A]$=false

ここで s はいかなる値も定まらない未定義状態⊥を含むとする。ある k 以降がすべて未定義の状態系列は、有限の系列と同一視する。論理式Aが状態系列 s 上で充足されるとは、⊥の状態を除いたある s 上でのAの解釈がtrue になることである。また、$\alpha_s[A]$ = trueを $s\vdash A$ と表わす。すべての状態系列で充足されるとき、**妥当な論理式**と呼ぶ。

4．2 様相 Horn 節

また、一般にnext やalways 演算子をもった様相述語論理体系は不完全な体系であることが知られており[4]、かつその部分システムである程度満足できる処理が行えることより、ここでは一階述語論理の Horn 節と類似の様相 Horn 節を導入する。回路実現言語はこの部分クラスとして特性化される。

定義4.1 (1) 原子様相論理式，MAF (Modal Atomic Formula)，とは、様相演算子で束縛された原子論理式 Modality A である。
(2) 様相論理式の連言形 ,CMF(Conjunctive Modal Formula)，とは 次の形の論理式である：
1)原子様相論理式の連言，
2)様相演算子で束縛された CMF，
3)CMF の連言。
(3) 様相 Horn 節 MHC(Modal Horn Clause) とは、つぎの形の論理式である。

$$MAF \leftarrow MAF_1,MAF_2,\ldots,MAF_n$$

ここで記号”，”は、連言（∧）を表わす。特に、MAF = φ の時（$\leftarrow MAF_1,MAF_2,\ldots,MAF_n$）、これをゴール、それ以外のものを事実（fact)と呼ぶ。

4．3 回路実現言語とその意味

回路とその表現の関係について前章で述べた。その回路実現の様相述語論理を用いた意味を忠実に反映した回路実現言語の設計について述べる。まず、基本ループに対応する

□(fun(g,X,Y) ∧ fun(reg,Y,○X)) = □(fun(reg & g,X,○X))

を考える。これを **{fun(g,X,○X)}** と表わすことにする。

定義 4.2 あるMHC（様相ホーン節）をAとする。Aの変数Xの充足系列とは、Aを充足するXへの代入値の系列で、これを sat(X) で表わす。

命題 4.1 MHC {fun(g,X,○X)} の X における充足系列は sat(X) = < x,g(x),g^2(x),...,g^n(x),...> となる。 但し、x ∈D は初期値 α_0[X]とする。

更に、条件式 p(X) をもつ標準回路は 条件 p(X) が満たされるまで g(X) を計算し、満たされた時端子Yに出力する回路である。これを **{fun(g,X,○X)}@p(X)** と表わす。

命題 4.2 MHC {fun(g,X,○X)}@p(X),fun(out,X,Y) のX，Yにおける充足系列は、次のように与えられる。 sat(X) = < x,g(x),g^2(x),...,g^{n-1}(x)>, sat(Y) = < ⊥,⊥,⊥,⊥,....,⊥, g^n(x)>. 但し、p(g^n(x)) = true とする。

様相ホーン節集合で、回路実現の次のような対応で現れる構文に制限したものを**回路実現言語**と呼ぶ。

Y = f(X_1,X_2,...,X_n) <==> fun(f,X_1,X_2,...,X_n,Y)
Y = p(X_1,X_2,...,X_n) <==> pre(p,X_1,X_2,...,X_n,Y)
[○X = f(X)] <==> {fun(f,X,○X)}
[○X = f(X)]@p(X) <==> {fun(f,X,○X)}@p(X)

定理 4.1 次の性質は等価である：

1) F(X) <= p(X)→f(X),F(g(X)) の X に初期値 x を与えたときの計算列 s(X).
2) FN 基本回路 [○X = f(X)]@p(X),out =f(X) の X に初期値 x を与えたときの計算列。
3) {fun(g,X,○X)}@p(X)の X に初期値 x を与えたときの X における充足系列。

この定理は回路実現言語を、様相述語論理に基づいた言語によって意味を正しく保存しながら実現出来ることを示している。

5．定理証明による自動合成

論理の枠組みで用いられる導出原理に基づく定理証明手法が，回路の自動合成にどのように応用出来るかその基本的考え方について述べる。

5．1 導出原理に基づく推論

述語論理による定理証明手法は、導出原理（Resolution Principle）またはその変形に基づいている。おおまかに言えば導出原理とはつぎのような論理式の簡略規則である。

導出原理：　～ＡＵＢ かつＡＵＣ　ならばＣＵＢ

様相述語論理については、まだ一般的な定理証明手法は確立されていない。本稿の様相述語の構文は前述のように制限されていることから、様相論理式を一階述語論理式へ変換することによって一階述語論理の導出原理を適用することができる。即ち、ここで提案する言語の構文はPrologの確定節と同様な A:-B1,B2,...,Bnの形をしている。そこでA=□A',Bi=□Bi'である。一般に、次の関係 □W→W ∧○□W が成立っている。ここでWは任意の述語論理式。この性質から、基本的には次のように展開する：

（現在の状態に関する論理式）∧（未来の状態に関する論理式）

定義 5.1 MHC のコンパイレーション C とは次のような変換である。

1) C(A) = A,　　2) C((A,B)) = {C(A),C(B)},
3) C(□A) = { A, □A },　　4) C(□(A,B)) = { C(□A),C(□B)}
5) C(□(A:-B)) = { A:- B ,□(A:- B)}　　6) C({A}@B) = { A , {A}@B }

命題5.1　論理式 {fun(g,X,○X)}@p(X)は次のように展開される：
{fun(g,X,○X)}@p(X):- fun(g,X,○X), {fun(g,X,○X)}@p(X)

コンパイレションによって、様相述語論理式は現在に関する一階述語論理式と○演算子に支配された未来に関する部分へ展開される。ここで採用した導出アルゴリズムは、現在と未来に関して展開した論理式の、現在に関する論理式につぎ次に導出原理を適用することに相当する。

５．２ 定理証明と自動合成

回路の基本素子や合成規則が様相ホーン節（MHC)の集合として与えられているとする。いま回路の構成に関する知識を表わす節集合を P= {p1,p2,...,pn}、実現したい回路の論理による表現に対応する任意のアトムを Qとする。このとき P から Q が**証明される**とは、P → Q が恒真なることである。Q が証明可能なら回路 Q は基本素子 P を用いて実現できることに対応する。この恒真性は前節で述べたように一階述語論理と同様に**反駁法**を用いて**導出原理**に基づいたアルゴリズムで一般的に証明することができる。従って、回路実現言語の意味は Prologのような論理型言語を用いてシミュレート出来る。ただし様相論理での導出は時間に関して論理式を展開しており、これは一階述語論理式では状態をもつ変数の値の受け渡しに相当している。そのためいまのところ assert や retract などの高階の述語を用いて実現している。この 定理証明過程であるトレース情報 が回路素子間の接続情報を表わしている。ただ、ループを含む回路の導出過程から回路接続情報を示すトレース情報だけを取り出すには、導出は時間について展開して実行されるので工夫する必要がある。現在のところ回路実現言語で記述した回路は、具体的データを入力すると回路動作をシミュレートしながら計算し、不定値（ある変数）を入力すれば接続情報を出力するようにしている。次に簡単な例を示す。

例 5.1 回路実現言語による回路の構造と動作記述

回路実現言語で記述したプログラム kairoは、add と mpy のループ結合回路の記述であり、X （不定元）を与えると回路接続情報を出力しval(X)である値 X を与えると計算値を出力する機能を持っている。

```
：-op(500,xfy).
kairo(X):-{f(X,next(X))}@p(X).
{f(X,next(X))}@p(X):- val(X),
  p(X)-> f(X,Y),(nonvar(X)->{f(Y,next(Y))}@p(Y));
  write('Y='), write(X).
P(X):-nonvar(X),X<20;var(X).
f(X,Y):-add(X,Z),write(add(X,Z)),nl,
        mpy(2,Z,Y),write(mpy(2,Z,Y)),nl,
  retract(val(X)),assert(val(Y)).

add(X,Z):-nonvar(X)->Z is X +1;true.
mpy(2,Z,Y):-nonvar(Z) ->Y is Z*2;true.
```

接続情報の取り出し

```
?-kairo(X).
  add(X,Z_17)
  mpy(2,Z_17,Y_2)
  no
```

値の計算

```
?-assert(val(1)).
?-kairo(X).
add(1,2) mpy(2,2,4).
add(4,5) mpy(2,5,10).
add(10,11) mpy(2,11,22)
Y=22  X=1;
```

５．３ 知識表現

これまで、回路素子や合成規則を公理として与えれば回路自動合成が出来ることを理論的に示した。本章では、前章で定義した回路実現言語で回路や変換の情報をどのように蓄積し利用するかについて説明する。これをここでは知識表現とよぶ。

5.3.1 表現の階層構造

表現の分り易さや処理の効率化のため 回路にはスキーマを最高位とし素子記述を最低位とする階層的記述を用いる。高位レベル素子はいくつかの素子の組合せであるマクロ素子として定義される。又、素子表現には関数の種類、述語、あるいはスタック、カウンタといった類別を行い、処理の効率化を図る。更に、一般的に成立つ規則、例えば直列・並列合成、適用規則、等はメタ規則として与え効率化とともに能力の向上を図る。

1) スキーマレベル：再帰方程式に対応する表現。

スキーマは述語schema で表わし、パラメータとして回路構成関数、変数等をとる。

schema(N,{P},{F},{X},{Y})

N:schema の型番号,{P}:述語の集合, {F}:関数の集合 ,{X}:入力の集合, {Y}:出力の集合

2) net レベル：ループを含む回路 ブロックを 単位とするレベルで次の構文で記述する。

net (<関数名>,<ﾊﾟﾗﾒｰﾀ>)) .

例えば、net(loop,P,G,X,Y) は {G,X}@P(X),out=Y に対応する基本ループ回路のマクロ表現である。

3) cmp レベル： 合成関数やマクロ素子を単位とするレベルで次の構文で記述する。

cmp (<関数名>,<ﾊﾟﾗﾒｰﾀ>).

例えば、cmp(mux,P,X,Y,Z). は mux 素子に対応する成分を示す。

4) 基本素子レベル：基本要素となる素子のレベルで論理系の公理に相当する。
データ領域Dおよびそこで定義される項、アトム、リテラル等は次の形式のデータとして与えられる：

a)定数 $c \in \Omega$ に対して const(c),
b)定数関数 $y = c(X) \in \Omega$ に対して c(X,Y),
c)関数 $y = f(X1,X2,...,Xn) \in \Omega$ に対して f(X1,X2,...,Xn,Y).
d)述語 $y = p(X1,X2,...,Xn) \in \Omega$ に対して p(X1,X2,...,Xn,Y).

基本素子は、これらの関数を項として含む述語として定義される。reg 素子はこの考え方を用いるとfun(reg,X,○Y) と表わされる。素子の種類に応じて fun,pre,map , const 等異なった述語で表現する。この述語の区別によって素子の類別を行なう。これをうまく使えば探索空間が指定出来、処理の効率化を図ることができる。一般的構文は次の通りである：

<分類述語名>（素子名,<入力変数>,<出力変数>）

5.2.2 合成規則の表現

(1) マクロ素子

幾つかの素子の組合せよりなる基本的な回路はマクロ素子として登録しておくと、記述が簡潔になるだけでなく修正や検証にも有効である。マクロ素子は直列・並列合成の組合せとして一般に与えられ、次の構文を用いる。

cmp(<種類>,<マクロ名>,<入力>,<出力>):-
cmp(<種類>,<関数直・並列合成表現>,<入力>,<出力>)

例えば、f_1、f_2 の直列、並列合成であるマクロ素子 f はそれぞれ次のように表される。

cmp(fun,f,X,Y):- cmp(fun,[f_1 & f_2],X,Y).
cmp(fun,f,X,Y):- cmp(fun,[f_1 # f_2],X,Y).

(2) 素子の直・並列合成に関するメタ規則

素子の組合せとして与えられた回路表現は、分解規則によって基本素子のレベルまで変換される。これは関数を変数とするメタ規則として与えることにする。例えば
直列・並列合成に対するメタ規則は次のようになる：

```
cmp(fun,[F & G],X,Y):-cmp(fun,F,X,Z),cmp(fun,G,Z,Y).
cmp(fun,[F # G],X,Y):-cmp(fun,F,X,Y1),cmp(fun,G,Z,Y2),Y = [Y1, Y2 ]
```

回路はこのcmpレベルでできるだけ分解するものとする。従って、このレベルで使用可能な素子を用いた表現が得られないときは、回路は実現不可能になる。

５．３ 変換規則

レベル間の変換は、それぞれの属性が継承されるように高レベルから低レベルへ行う。まず一般的記述の構文はつぎの通りである。

```
schema(N,<パラメータ>) :- {net記述の集合}
net(<回路名>, <パラメータ>) :-{ cmp 記述の集合}
cmp(<素子名>,<パラメータ>) :- {素子記述の集合}
```

例えば 1章で述べた11種の schema について、schema 間の変換規則[2]をこの構文で与えている。その一部を示すと次のようである：

```
schema(3,P,F,H,G,X,Y):-schema(2,[P&π1[P&π1]],[π2#F&π1[π2]],
                                [G&π1#π2[G&π1#H&π2]],X,Y).
schema(6,P,F,H,G,X,Y):-schema(1,P&π1,F,G&π1#H,X,Y).
schema(7,P,F,H,G2,G3,X,Y):-schema(1,P&π1,F,G2&π1#H&[π2#G3&π1],X,Y).
```

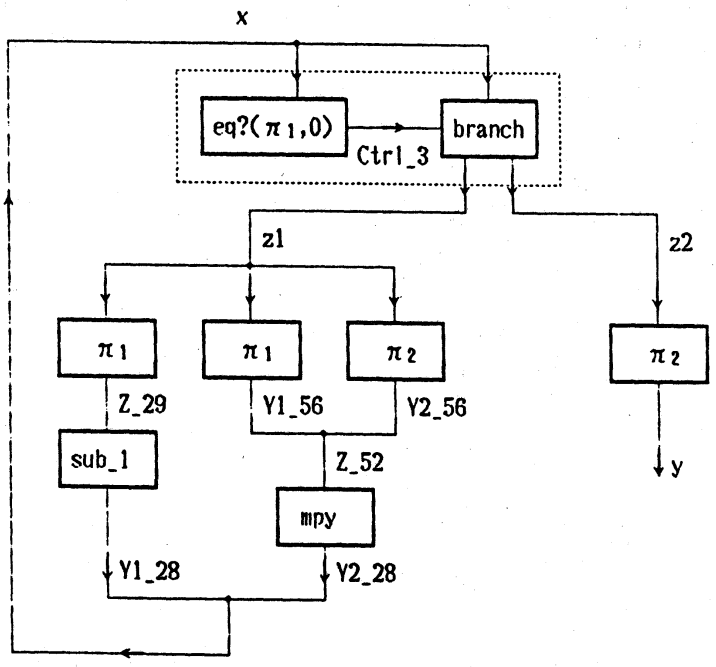
5．4 合成例

現在PROLOG-KABAを用いて以上の考えに基づいたシステムのプロトタイプを構成している。実現システムでは1章で述べた１１種の schema について回路合成が出来るようにしている。次にこのシステムを用いた簡単な合成例を示す。ここで sub_1 と mpy には遅延 reg を含めて描いている。

```
Factorial 関数
| ?-schema(1,eq(pai1,0),
|                pai2,
|                sub_1&pai1#mpy&[pai1#pai2],
|                x,y).
[pre(eq(pai1,0)),x,Ctrl_3]
    [fun(branch),[Ctrl_3,x],[z1,z2]]
```

```
[fun(pai1),z1,Z_29]
[fun(sub_1),Z_29,Y1_28]
[fun(pai1),z1,Y1_56]
[fun(pai2),z1,Y2_56]
    [Y1_56,Y2_56]==>Z_52
[fun(mpy),Z_52,Y2_28]
    [Y1_28,Y2_28]==>x

[fun(pai2),z2,y]
```



6．結論

本論文では、高レベル記述からの回路の自動合成問題を人工知能的観点から考察した。高レベル仕様記述言語として再帰方程式を採用し、回路を直接表現する回路実現言語を様相論理を用いて設計した。回路自動合成は、その再帰方程式から回路実現言語への変換として定式化され、その枠組みで原理的に回路の自動合成が行える事を理論的かつシステムのプロトタイプを構成することによって示した。そこでは、基本的手法として新しく様相述語論理に基づいた定理証明の手法を用いている。そこでは、回路素子や変換規則の記述を簡潔にし、処理効率を上げるため、素子の表現に階層性を導入し、変換規則のメタレベル記述等推論機構の工夫を行っている。最後に、Prolog-KABAを用いたシステムのプロトタイプを構成し、この手法の有効性を明らかにした。そこでは、様相述語論理に基づいた回路実現言語の意味を正しく反映する記述言語を設計し、回路動作の正しさの検証と回路構成が共に行えるように工夫している。残された課題としては、処理の組合せ的爆発にどう対処するかと関連して、知識の表現やメタレベルの変換規則・制御規則等の問題がある。また、理論的にはスキーマ変換の特性化がある。これには、高階の論理やタイプ理論等を用いて更に検討する予定である。また、スタック等複雑なデータ構造を有する素子を用いた場合の回路構成法の問題等も残されている。

謝辞：プログラムの作成や資料の整理に手伝って戴いた研究室の安孫子、阿部両君に感謝します。また、この研究の一部は、科学研究費一般(C)60580016の援助の下に行われたものです。

文献

[1]原尾、岩沼：定理証明的手法を用いた回路の自動合成,数解研講究録 591,pp65-73,1986.5
[2]原尾、岩沼：定理証明的手法による再帰方程式から回路の自動合成,信学技報ｺﾝﾋﾟｭﾃｰｼｮﾝ CMP86-75,pp23-34,1987.2,
[3]G.Huet and B.Lang: Proving and Applying Program Transformations Expressed with Second-Order Patterns,Acta Informatica,11,pp31-55,1978
[4]岩沼、原尾、野口：時間と空間を扱う様相述語論理の不完全性とその相対的完全性、信学技報ｺﾝﾋﾟｭﾃｰｼｮﾝ,COMP86-29,pp11-19, 1986.9
[5]S.D.Jhonson: Synthesis of Digital Designs from Recursion Equations, ACM distinguished Dissertations, MIT press,1983.
[6]F.D.Cerro:MOLOG:A System That EXtends Prolog with Modal Logic,New Generation Computing,Vol.4, pp35-50,1986.
[7] Z.Manna : Mathematical Theory of Computation, MacGraw Hill Inc.1974
[8] B.C.Moszkowski : Executing Temporal Logic Programs,Cambridge Univ. Press,1986
[9] N.J.Nilsson: Principles of Artificial Intelligence,Tioga Publishing Co.1980
[10]上原：ＣＡＤにおけるProlog,情報処理, Vol.25 No.12,1984.
[11]J.D.Ullmann:Computational Aspects of VLSI,Computer Science Press,1984
[12]S.A.Walker and H.R.Strong:Characterizations of Flowchartable Recursions,JCSS 7, pp404-447,1979.
[13]W.S.Wojciechowski,A.S.Wojcik: Automated Design of Multiple-valued Logic Circuits by Automatic Theorem Proving Techniques,IEEE Trans.on Comput.Vol.C-32,No.9,1983.